Section	Requirement	Description	Priority	Rationale
1.1	**Facade Pattern Enforcement**	Centralizes all major system operations through SystemFacade to provide a single, controlled entry point for the application.	1	All high-level operations (login, logout, submitAnswer) must be centralized through SystemFacade to manage interactions between system components.
1.2	**Singleton Enforcement**	Restricts SystemFacade, UserList, and QuestionList to a single instance via getInstance() to ensure global state consistency.	1	SystemFacade, UserList, and QuestionList must implement the Singleton pattern via getInstance() to ensure a single centralized state.
1.3	**Data Management**	Utilizes DataLoader and DataWriter to manage the initialization and persistence of all user and question data.	1	SystemFacade uses DataLoader to initialize data and DataWriter to persist Users, InterviewQuestions, and QuestionSubmissions.
2.1	**Account Structure**	**Stores core user data including a unique UUID, email, passwordHash, and name to identify and manage accounts.**	1	Users are identified by a UUID and must store email, passwordHash, name, and login timestamps.
2.2	**Internal Password Validation**	**Employs private helper methods within the User class to verify that passwords meet specific complexity requirements.**	1	The User class must contain private helper methods: hasUpCase, hasLowerCase, hasDigit, and hasSpecialChar.
2.3	**Authentication Logic**	**Delegates user credential verification to the UserList.authenticate() method to manage secure access.**	1	UserList handles core authentication via authenticate(), while SystemFacade provides the high-level login() entry point.
2.4	**Role-Based Access**	**Uses boolean flags isAdmin and isContributor to differentiate permissions and restrict administrative functions.**	1	Privileges are determined by isAdmin and isContributor boolean flags within the User class.
3.1	**Profile Association**	**Connects each User to a Profile containing their academic background and contact information.**	1	Every User has an associated Profile containing academic details like school, major, and graduation year.
3.2	**Profile Management**	**Facilitates updates to a user's school, major, and graduation year through the updateProfile() method.**	2	Profile.updateProfile() modifies academic details, and updateUpvotes() tracks community engagement.
3.3	**Rank & Progress**	**Calculates the user's current standing via getRank() and tracks their overall advancement with getProgress().**	3	Profile.getRank() returns a Rank enum, and User.getProgress() returns a Progress object for tracking.
4.1	**Question Definition**	**Defines an InterviewQuestion by its difficulty, category, type, and specific content details.**	1	InterviewQuestion must include difficulty, type, category, and an author reference.
4.2	**Question Management**	**Uses a HashMap within QuestionList to store and manage questions by their unique UUID.**	1	QuestionList manages the collection using a HashMap keyed by UUID for efficient retrieval.
4.3	**Filtering & Search**	**Provides specialized methods to retrieve lists of questions filtered by their specific Category or Difficulty level.**	1	QuestionList provides filtering via getByCategory() and getByDifficulty().
5.1	**Answer Submission**	**Enables users to submit code-based solutions through the SystemFacade.submitAnswer() operation.**	1	SystemFacade.submitAnswer() creates an Answer using the specific User and Question IDs.
5.2	**Interaction Logic**	**Implements voting capabilities in the Answer and Comment classes to track community sentiment via upvote() and downvote().**	2	Both Answer and Comment classes must support upvote(), downvote(), and getVoteScore().
5.3	**Comment Metadata**	**Records the author, timestamp, and isEdited status for every user-generated comment.**	2	Comment objects must track the isEdited status and the author's name.
6.1	**Category Enum**	**Organizes technical content into specific domains like BIG_O, ARRAY, and LINKED_LIST.**	1	Must include categories such as BIG_O, ARRAY, and LINKED_LIST.
6.2	**Difficulty Enum**	**Classifies the complexity of all interview questions as EASY, MEDIUM, or HARD.**	1	Must include difficulty levels: EASY, MEDIUM, and HARD.
6.3	**QuestionType Enum**	**Specifies the required response format, such as CODING, MULTIPLE_CHOICE, or SHORT_ANSWER.**	1	Must include types: CODING, MULTIPLE_CHOICE, and SHORT_ANSWER.